Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 67 (2015) 312 – 321

Procedia Computer Science

q6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Infoexclusion (DSAI 2015)

Low-cost natural interface based on head movements

João M. S. Martins[a,]*, João M. F. Rodrigues[a,b], Jaime A. C. Martins[b]

[a]*Instituto Superior de Engenharia, University of the Algarve, 8005-139 Faro, Portugal*
[b]*LARSyS (ISR-Lisbon) and CIAC, University of the Algarve, 8005-139 Faro, Portugal*

Abstract

Sometimes people look for freedom in the virtual world. However, not all have the possibility to interact with a computer in the same way. Nowadays, almost every job requires interaction with computerized systems, so people with physical impairments do not have the same freedom to control a mouse, a keyboard or a touchscreen. In the last years, some of the government programs to help people with reduced mobility suffered a lot with the global economic crisis and some of those programs were even cut down to reduce costs. This paper focuses on the development of a touchless human-computer interface, which allows anyone to control a computer without using a keyboard, mouse or touchscreen. By reusing Microsoft Kinect sensors from old videogames consoles, a cost-reduced, easy to use, and open-source interface was developed, allowing control of a computer using only the head, eyes or mouth movements, with the possibility of complementary sound commands. There are already available similar commercial solutions, but they are so expensive that their price tends to be a real obstacle in their purchase; on the other hand, free solutions usually do not offer the freedom that people with reduced mobility need. The present solution tries to address these drawbacks.

Peer-review under responsibility of organizing committee of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI 2015)

Keywords: Accessibility; Assistive Technologies; e-Inclusion; Face Tracking; Human-Computer Interaction; Kinect sensor.

* Corresponding author. Tel.: +351 964935902
E-mail address: joao.miguel.martins@sapo.pt

Peer-review under responsibility of organizing committee of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI 2015)
doi:10.1016/j.procs.2015.09.275

1. Introduction

People often seek freedom through virtual means, or they simply need to use a computer to work. However, not everyone has the skills to interact with a computer in the same way. People with motor difficulties or reduced mobility, including seniors, ultimately do not have the same freedom as other persons to use a mouse, keyboard or touchscreen. Since electronic devices are all scattered through different occupational sectors, the need to interact with a computer is a crucial capacity. Nowadays, with all the economic problems that families and businesses support, including government cuts in accessibility programs, it is imperative to develop low cost human-computer interfaces (HCI). These low cost interfaces can be developed, for instance, by reusing outdated sensors from game consoles, such as the Microsoft Kinect.

We can find in the literature many techniques and algorithms for the detection of faces[1,2], facial features[3,4], and facial poses[5,6]. However, we are most interested in analyzing complete functional systems that perform the same or similar functionalities as the solution we intend to develop, i.e., a natural interface for persons with reduced mobility to interact with a standard computer based on head or/and facial movements.

Chen *et al.*[7], developed a system that allows a user to control the mouse pointer by moving his head, still requiring a positional sensor attached to it. Rajaee[8] developed in 2004 a project that allowed people with physical disabilities access to the internet using a switch on a computer or a Personal Digital Assistant (PDA). Through specific software, one could navigate and send emails using switches. Connor *et al.*[9,10] developed the Camera Mouse, as a substitute to the conventional mouse, especially for users with motor difficulties. It is a software that works with any conventional camera connected to the computer, and is based on face tracking. The commercial product DynaVox Eyemax System[11,12], is also a system for people with reduced mobility and is controlled only by eye movements. This software allows interaction using the strategy that best suits the user (symbols, photographs, words and/or letters). It has several high quality synthesized voices, and supports Bluetooth and Wi-Fi for easy connection to the Internet. It can even use a configurable infrared emitter, which allows basic control of television, radio, air conditioning or any other devices. Another commercial solution is the Quha Zono Mouse[13]. It's a plug and play solution, wireless, allowing through the user's body movement, to move the cursor on screen, select, click and close items. The device can be attached to the head, wrist, foot or any other body part. Visual Interaction GmbH offers the myGaze Assistive Eye Tracker solution (also a commercial solution). This system is adapted to be used with screens from 10" to 22". The sensor is placed in front of the monitor, and connects to the computer via USB cable[14]. The manufacturer also offers a development package, consisting of an API (Application Programming Interface) and a sensor. With the API, this sensor can be integrated into different Operating Systems (OS). Xcessity Software Solutions developed KinesicMouse, which uses the Kinect sensor to simulate the use of a conventional computer mouse, allowing single and double clicks with certain facial movements. This solution does not require any sensor or structure fixed on the user's body, which relays a freedom of movement, providing that the user is in the sensor's field of view. This software (paid license) is available for download at the KinesicMouse site[15]. If we compare prices, the last one is the cheapest, and in terms of computer mouse operation, sufficient. It is however, a more expensive solution than freeware/open source ones based on a cheap webcam, but allows greater freedom of movement thanks to the depth measurements provided by the Kinect sensor.

This paper focuses mainly on the development of a low-cost interface that allows control of the computer without a keyboard, mouse or touchscreen, while being easy to use and open source, so that those who have the possibility and wish, can improve or adapt to their liking. The interface is based on the Kinect sensor, and allows computer control only by the movement of the head, eyebrows (up/down) and the mouth (opening/closing), in addition to available voice commands. A final full-functional solution that can work in any type of monitor is presented. The main contribution is the development of a fully functional, ready to use, low cost natural mouse interface, that is being offered as open source software, which the user (with the help of a programmer) can improve and adapt to his/her liking. The proposed solution was tested with real users.

It is also important to mention that systems similar (with the same principles) to the one presented have a larger application. For instance in the present, and also in a near future, systems similar to this one, that can track the user face and/or eyes can be integrated in different applications such as interactive installations, or virtual public relations, which interact in real time with the user, by knowing their head and/or body positions, see e.g., the PRHOLO project[16,17].

2. The framework

As mentioned in the previous section, the system has only one hardware component in addition to any standard computer (with a Microsoft Windows Operating System installed), which is a 1st generation Kinect sensor that can be bought in any second hand store with prices around 70€. In terms of software, the application should allow the user to control the mouse cursor and respective clicks, while also having the system setup and configuration through easy-to-use menus. The developed application was named "UserTracking", and was tested with the Kinect "for Xbox 360" and "for Windows" sensors, both using a resolution of 640×480 px (pixels) for RGB and depth, and a frame rate of 30 fps (frames per second) maximum. For the application development, it was used: for the application development, it was used the Kinect for Windows SDK v1.8 (Software Development Kit) available from Microsoft[18] and the C# programming language[19].

The audio acquisition, RGB, and depth data was done by a Kinect sensor placed above the screen, centered with it (see Fig. 3 and 4). From the depth and RGB information, the closest user is detected and the face position is estimated. The Animation Units[18] are also extracted, for the eyebrows and mouth, which will in turn serve as the "mouse buttons" (clicks). Sound, which serves as additional (optional) commands for the "mouse buttons", is acquired in parallel. Having covered the mouse buttons (see Sections 2.2 and 2.3), it is first important to know the exact position of the mouse in the monitor, which is achieved by estimating the face pose, i.e., where the face is looking, to the left/right or up/down, and move the mouse accordingly.

2.1. Mouse position controlled by the movement of the face

User detection is performed by the *FaceTracking* algorithm[20] using the Kinect for Windows SDK v1.8, which requires data (images) obtained simultaneously by the RGB and IR (depth) cameras. In addition to those images, it also requires information of the skeleton, so that it can reduce the image size, namely the region of interest where the detection and tracking algorithms are applied. Having a face detected, the next step is to calculate the "focal point" (the point where the face is looking at, in the screen).

First, to determine the focal point, we must know where the users face is, in respect to the Kinect position, so we can determine the spatial location of the face relative to the Kinect, and the screen location in relation to the Kinect sensor. In the Kinect reference space, all points that represent a face obey the Cartesian system shown in Fig. 1 (left). The reference origin is physically in the midpoint between the RGB camera lens and IR receiver. When a face is detected by the Kinect, the Active Appearance Model algorithm (AAM)[21,22,23] applies the CANDIDE-3 facial model[24].

An example of a user with the face points represented by the CANDIDE-3 parameterized face mask is shown in Fig. 1 (middle), with the red arrow (vector) representing the orientation of the face relative to the Kinect. As default, we chose this vector originating in the midpoint between the user's eyes. In Fig. 1 (right), the direction the user is facing is represented by vector $\vec{V}$ (corresponding to the direction of the arrow in Fig. 1 middle) and the Kinect's reference origin is O. Point P is the midpoint between the user's eyes (and therefore the origin of vector $\vec{V}$). Point Q represents the focal point defined by the intersection between vector $\vec{V}$ and the plane Π formed by the display.

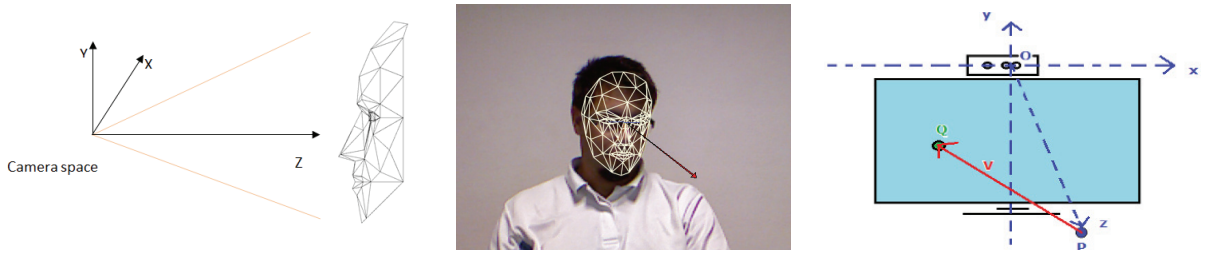

Fig. 1. Left: face mesh representation in Kinect space. Middle: facial representation using the CANDIDE-3 model and respective $\vec{V}$ (red) orientation vector. Right: representation of the user's face coordinates by the centroid position and orientation vector $\vec{V}$.

As the Kinect is placed on the top-center of the monitor (see Fig. 3 and 4), the integrated sensors lie on the screen plane, so points like Q, with Cartesian coordinates (x, y, z) in the screen Π plane have a null depth component, i.e., z=0.

Initially, the coordinates of the focus point are determined relative to the Kinect in meters (m), and then they are mapped to image coordinates on the screen in pixels (px). This is mathematically formulated as an interception of a ray with an object, which corresponds to the coordinates of point Q, that is given by $Q = P + t\overrightarrow{V_o}$. Considering $P = (x_o, y_o, z_o)$, and $\vec{V} = (x_d, y_d, z_d)$ we obtain $Q = (x_q, y_q, z_q) = (x_o + x_d t, y_o + y_d t, z_o + z_d t)$. Since Q belongs to the same plane $z = 0$, we can now conclude $z_o + z_d t = 0 <=> t = -z_o/z_d$, and therefore the point Q can be written as $(x_q, y_q, z_q) = (x_o - (z_o/z_d)x_d, y_o - (z_o/z_d)y_d, 0)$.

Note that when the user is parallel facing the screen ($\vec{V}//\Pi$), $z_d = 0$. As the API only tracks faces when the deviation angle (yaw angle) is between [-45º, 45º], and when the elevation angle (pitch angle) is between [-20º, 20º][7], then, $z_d < 0$. It is important to notice, that while the API can provide the centroid, yaw and pitch angles, it does not provide the orientation of vector $\vec{V}$.

Vector $\vec{V}$ is quite simple to compute also by applying trigonometric functions, nevertheless it can consume more computational resources (particularly in older computers), so it was chosen to adopt another strategy. We used points of the facial mask returned by the Kinect API (as mentioned above, the API uses the facial model CANDIDE-3, which is shown in Fig. 2, left). The origin point P of the orientation vector ($\vec{V}$) was determined from the eyes (Fig. 2, middle). For each eye, we selected three points which form a triangle. In the case of the right eye, the triangle ABC, for the left eye, the triangle DEF.

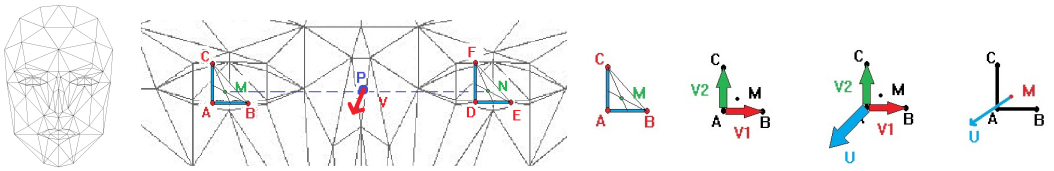

Fig. 2. Left: facial model CANDIDE-3 (adapted from[24]). Middle: computation of P, the origin point of the orientation vector ($\vec{V}$). Right: the orthogonal vector for the right eye.

The centroid, Fig. 2 (middle), of ABC and DEF triangles are represented by points M and N respectively, and can be calculated as $M_{x,y,z} = (A_{x,y,z} + B_{x,y,z} + C_{x,y,z})/3$ and $N_{x,y,z} = (D_{x,y,z} + E_{x,y,z} + F_{x,y,z})/3$. Since P is the midpoint between the two centroids it can be determined by $P = (x_o, y_o, z_o) = [M_{x,y,z} + N_{x,y,z}]/2$. To compute the orientation vector $\vec{V}$, it was necessary to compute the two vectors $\vec{U}$ and $\vec{W}$ each orthogonal to the plane defined by point {A, B, C} and {D, E, F} (right and left eye) respectively. Fig. 2 (right), illustrates this for the right eye, as $\vec{V}_1 = \overrightarrow{AB}$, $\vec{V}_2 = \overrightarrow{AC}$, and $\vec{U}$ is obtained by the cross product, $\vec{U} = \vec{V}_1 \times \vec{V}_2$. We can apply the same reasoning to compute $\vec{W} = \vec{V}_3 \times \vec{V}_4$, the orthogonal vector to the left eye, with $\vec{V}_3 = \overrightarrow{DE}$, $\vec{V}_4 = \overrightarrow{DF}$. Finally, one can calculate the orientation vector $\vec{V}$. This vector is actually the unit vector director of the line segment $\overline{PQ}$ defined $\vec{V} = (\vec{U} + \vec{W})/|\vec{U} + \vec{W}|$. The origin of the orientation vector $\vec{V}$ is located at point P, as shown in detail in Fig. 2 (middle).

So far all the calculations were conducted in spatial coordinates (3D) in meters, which need to be converted into screen/monitor coordinates (in px) for the mouse pointer. We define point G (in px) as the corresponding point Q (in m), now converted to the screen coordinate system in pixels. One should take into account that the two systems have the vertical axis in opposite directions, as shown in Fig. 3 (left).

For this conversion, we need to know the exact position of the Kinect sensor in relation to the monitor/screen. It is requested in the application's initial startup (just one time) for the person who will initially configure the application to manually enter the Kinect's position (in centimeters) relative to the monitor/screen (see also Section 3 and Fig. 4). We define w as the horizontal distance from the Kinect to the upper left corner of the monitor, h the distance from the base of the Kinect to the top of the visual area of the screen (Fig. 3 second column), m_w the width, and m_h height of the screen, r_w the horizontal resolution of the image, and r_h is the vertical resolution used in the screen. This allows for the translation of point Q with coordinates $(x_l, y_l, 0)$ (in m) to G with coordinates $(x, y, 0)$ (in px): $G_x = (x_l + w).r_w/m_w$, $G_y = (y_l + h + 0.05).r_h/m_h$, where the constant 0.05 reflects the 5 cm distance

between the lowest point of the Kinect base and the point of origin of the spatial coordinates of Kinect (Fig. 3, third column).

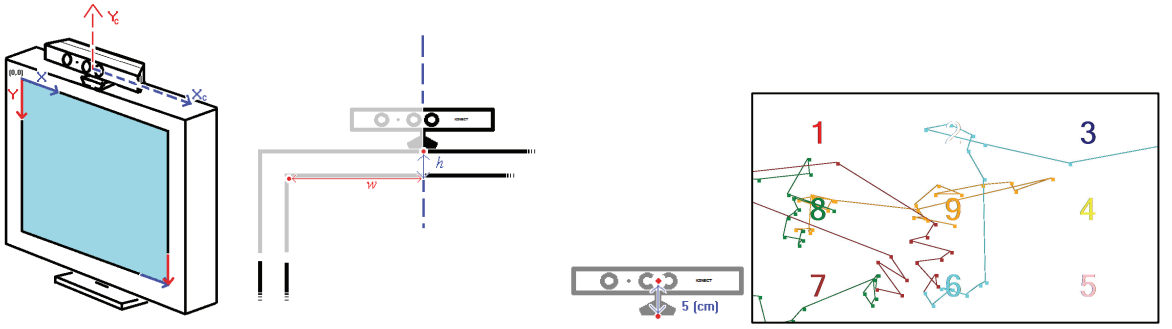

Fig. 3. Left to right: representation of the coordinate systems of the monitor/screen and the spatial coordinates of the Kinect system. The measures *w* and *h,* necessary for system configuration. The distance between the reference point and the Kinect base. An example of a calibration process.

Data acquisition from the Kinect is subject to noise, manifested as the mouse position floating when the user has no body or head movements. Some strategies were tested to reduce this noise: (a) the average of the *n* last mouse points, (b) the weighted average of the *n* last mouse points, (c) the average of the last *n-t* points (excluding *t*, the outliers), (d) the application of a Kalman filter, and (e) the development of a dead zone, i.e., an area where the mouse does not suffer (coordinate) changes, together with a delay in the mouse acceleration and velocity.

None of the strategies a) to d) presented sufficient stabilized results. The strategy e) was the one chosen due to be less CPU demanding, and still present good results. It was divided into two main components: (i) the definition of a dead zone, i.e., a circular area with a radius 3 px around the last mouse position, where the coordinates of the mouse are kept fixed even if the mouse coordinates are floating inside that area, for 3 s (seconds). Multiple values have been tested, and it was found that those allowed for comfortable use (nevertheless, they can be changed in the configuration menu of the application). With higher values the mouse coordinates on the screen are more stable, but lose accuracy for small intended movements, e.g., when it is pretended to access small icons (like virtual keyboard letters).

In combination with the previous strategy, it was developed (ii), a delay in the acceleration and displacement/velocity of the mouse on the screen, proportional to the RGB image acquisition rate. This delay factor is again adjustable at the beginning of the application; it also serves to "adjust" the user's comfort. Different users feel more or less comfortable with different accelerations and velocities on the mouse movement, as users with certain types of reduced mobility it is expected a preference for a slower moving pointer.

There is also an automatic calibration step performed in the first use of the software. During this process, the user should aim the face (for convenience, it was said to "point the nose") to the nine points (numbers) that appear sequentially on the screen, Fig. 3 (right). The user should try to stay at each point for about 3 seconds. After this time, another fixing point appears. And so on, up to nine points. The result of this process allows the user to have greater control of the mouse pointer. In the case presented (Fig. 3, right) the user was asked to repeat the process, once he/she missed two points (4 and 5).

2.2. Mouse buttons controlled by the movement of the face

As mentioned before, the mouse buttons can be simulated by using the mouth and eyebrows. In each acquired Kinect frame, it is extracted the Animation Units (AU)[18,25] of the eyebrows and mouth. The values of those AU change between 0 and 1. When the user moves his eyebrows up, and his AU reaches a certain threshold value and the software sends a command to the Operating System[19,26], simulating a mouse button click. When the eyebrows AU value is below the threshold value, the software sends a command simulating the release of the mouse button. The same procedure is carried out for the opening and closing of the mouth (which has its own AU). Those

thresholds (limits) can be adjusted manually by the user in the software's configuration menu. Typically, the lowest values are more likely to produce false positives, while higher values increase the difficulty to create true positives. In practice, different users can have different values that they feel comfortable. The most common values, were 0.2, 0.3 and 0.4 on the empiric usability tests (see Section 4).

2.3. Mouse actions controlled using voice commands

The development of voice commands is quite trivial, since it is sufficient to use the Kinect SDK, which integrates with the Microsoft Speech API (requires installation). First, a list of phrases or words is defined, which will be the recognized by the Microsoft Speech API. This list of commands is call a grammar. The structure of this grammar follows the XML format[27,28] and every voice command is associated with one action and simulates the same effect as a physical mouse click (the same as in the Section 2.2). So far, we only developed English voice-commands: *Left button*, *Right button*, *Click*, *Left click*, *Right click*, *Double click*, *Click down*, *Click up*, *Drag* and *Drop*. More languages can be used, as it is only necessary to rewrite the XML file to the language in question, and install the classes of the API necessary for that language.

3. The prototype

As already mentioned, the Kinect sensor must be mounted on top of the monitor, preferably in the center. If the monitor does not have enough space to ensure the stability of the sensor it should be put on top of a support, see e.g., Fig. 4 (top). After mounting the Kinect and running the application for the 1st time, the initial configuration menu appears (Fig. 4, bottom). This menu allows for the initial configurations, including inputting the distances of the Kinect to the left corner of the monitor and to the visible screen area (see Section 2.1). After the initial configuration, a smaller menu-bar is available, Fig. 5 (left), which keeps itself visible until a user has been detected. After user detection, Fig. 5 (right), the bar disappears after 30 s to reappear whenever the user takes the mouse pointer to the upper right corner, for some software control options. After these very easy-to-do procedures, the application is ready to work. If, by any circumstance, the user needs to temporarily disable the application, e.g., use the hand mouse, it is only necessary to put the application in "Parking Mode" (see Fig. 5) to disable the pointer control functionality. Pressing again the same menu-bar icon resumes the head control mode.

4. Results

Several users tested the developed system with the intention to assess the overall feasibility of the system. The tests had the goal to initially assess if the system was able to work correctly with users having different characteristics (face morphology, skin and eye color, glasses, hair, bear, etc.) on indoor environments with different lighting. For each test session, the users were presented with a list of tasks to be performed. Before each test session, each user did a very small training session, to adjust, calibrate and understand the system. During the test session, each user was filmed doing all the actions, thus, evaluating the workability of the system and assess the time he/she took to perform each task. At the end of each session, an inquiry was done to know the comfort level felt by the user, and his/her feedback including improvement suggestions.

Relevant user biometrics were collected in order to facilitate further analysis, including gender, age, skin, eye color, haircut, etc., including other facial accessories as well as computer literacy/experience. For the tasks carried during the test sessions, users had access to two virtual keyboards. In addition to the software's own native keyboard, it was also available Microsoft Windows' virtual keyboard. As mentioned, all procedures were timed, and were divided in "essential" and "non-essential" tasks. For the first, it was mandatory for the users to comply exactly with the instructions presented in the procedure to achieve the goal of the task, while for the non-essential they were free to choose the strategy used to achieve the goal.

In summary, the first task instructed the user to create a folder on the desktop with a specific name "Test N. #", where "#" is the sequence number of the test (Task 3.1). Then, they was asked to open a browser (Task 3.2), access the YouTube website, and play a specific song (Task 3.3, from 3.3.1 to 3.3.3). With those tasks, it was intended to test the usability of the virtual keyboard (and simultaneously check the system's response to voice commands in the

presence of music). Then, the user had to download a file (.zip) from a site in the Internet, and save it to a folder created on the desktop (Task. 3.4). After locally saved, the file should be extracted (Task 3.5, from 3.5.1 to 3.5.3). Task 3.6, from 3.6.1 to 3.6.4, users were asked to create a drawing of a face, and save it to a .jpg file. Finally, on Task 3.7, from 3.7.1 to 3.7.7, users were instructed to create a text document with a specific formatting, containing three phrases. These tasks were used to test whether the developed system allowed users to write a document using only the movements of their head and a virtual keyboard.

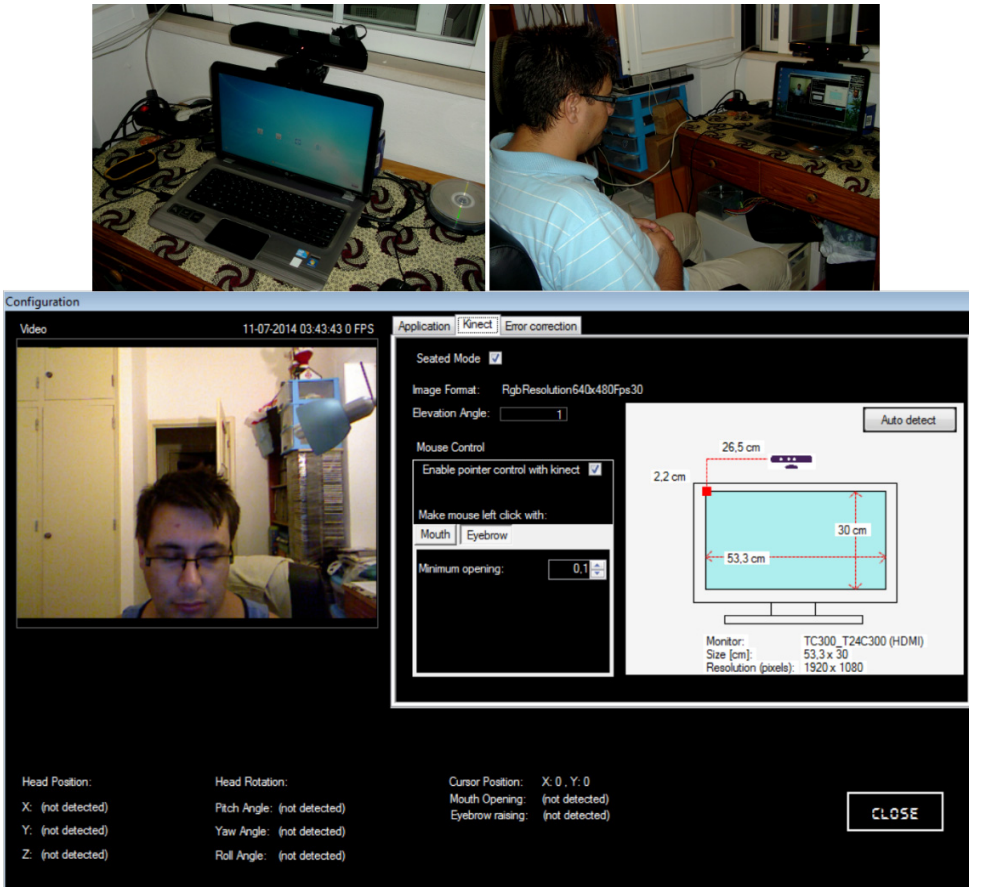

Fig. 4. Top, a scenario that shows how to position the Kinect over a (laptop) monitor and a user working with the system. Bottom, configuration window of the software.

All collected data were summarized in tables and graphs for comparison. Eight volunteers were recruited, of which three gave up during the calibration and training phase. The reasons for withdrawal were reported as: one was due to the difficulty to control the mouse pointer (it was too sensitive to the movements performed with the head), the other complained of great discomfort when using the system, including neck pain. The third one had bifocal lenses, and the fact he has to switch between the lens regions to see close/far created a great difficulty and discomfort during system calibration.

All users who performed the trials had already experience with the software necessary for the required tasks, i.e., they had already used a browser, downloaded files, watched videos, listened to music on YouTube and were familiar with Microsoft Paint and OpenOffice Writer or Microsoft Word. The time each user took to conduct the experiment was mostly unaffected by the lack of experience with the available software. Figure 6 left, shows the time each user took in each task.

Fig. 5. Left, the system when a user is not detected, and right, after a user being detected.

By observing the chart (Fig. 6 left), it is easy to conclude that all procedures involving the use of the virtual keyboard are the most time consuming. Tasks 3.3 and 3.4 usually took a significant amount of time due to two reasons: the fact that the software's compact keyboard became too impractical to write, forcing the users to choose the (virtual) keyboard from Microsoft Windows; the other reason was related with using the vertical scroll bars that appeared in the web browser. Task 3.6 is a true test to control the mouse pointer; in this task, the users had to draw a face with circles and lines in Microsoft Paint, forcing them to coordinate the displacement of the mouse with the clicks to control the Paint application. In this task, it was found that the worst time was due to the artistic nature of User 3. In addition, this user chose to start the test from this point, ignoring all previous goals, mentioning a lack of willingness to continue with the test. All invalid tasks are presented with a "zero" time, due to sub-tasks that the users decide not to do.

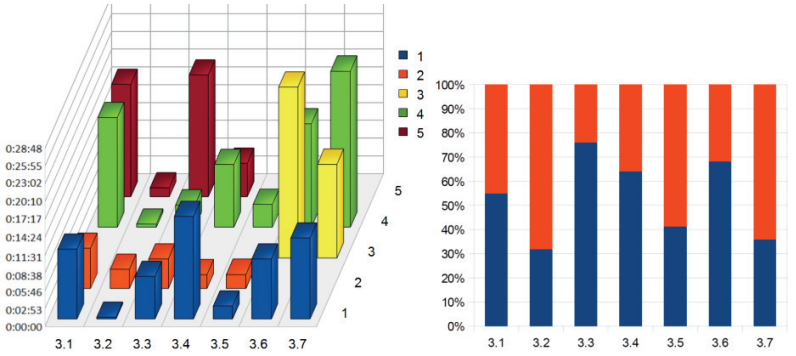

Fig. 6 Left, time spent in each test (see text). Right, comparison between time spent doing the tasks by users with glasses (in red) and without glasses (in blue).

According to some users, the system becomes simpler to control after some practice; also, some of them found that the mouse pointer was moving too fast for the movement they were doing with his/her head (a problem easily corrected in the configuration menu). Users also complained, from time to time, that clicks were happening without their order (false clicks), which was distressful; some also complained that voice commands and the mouth movements are not always recognized.

In terms of facial characteristics (skin, eye color, glasses, etc.), it was not noticed any relevant performance hit in the system. Although the very low number of participants does not allow for statistically significant conclusions, for example, in the glasses/no glasses case, the average time consumed to do the tasks was almost the same, see Fig. 6 (right), where the red bars show the percentage of time above the average for each task for a person with glasses and in blue without glasses. Moreover, users presented some suggestions to improve the native keyboard, such as the creation of a "www" or ".com" buttons. Some users suggested that the desktop should be divided into two parts: one

should be an area for windows, open documents, etc., and the other part should always have a visible keyboard; others suggested the elimination of the compact keyboard, and the exclusive use of the Microsoft Windows (virtual) keyboard.

5. Conclusions

We presented the development of a human-computer interface that allows for touchless computer control, avoiding the use of a keyboard, mouse or touchscreen. By reusing Microsoft Kinect sensors from old videogame consoles, the software was developed at a reduced hardware cost, while being easy to use and supporting an open source interface, which allows controlling the computer using only the head, eye or mouth movements, with complementary sound commands.

Initial, small-scale system testing was done so far with eight users, with the goal to understand which tasks or movements they had greater difficulty in performing. In addition, we also wanted to know the comfort level experienced during the use of the interface. The feedback from users was crucial, for this reason, and an open survey gave them freedom to suggest system improvements.

When using this type of head control system, we also found desirable modifying the OS desktop's visual theme, in function of user characteristics. For instance, for some users navigating and clicking in small icons generates great discomfort, resulting in neck pain, and the constant need to move away from the computer.

As expected, the tests revealed that the system could be used by people of light or dark skin, using glasses, beard, etc. However, it was found that the use of some types of glasses could bring discomfort to the user when using the application, such as glasses with bifocal lenses, due to their lens transitions. It was also noticed that the controlling the OS with the head requires some time to practice. It is therefore recommended, at least in the beginning, that the user does not abuse the time passed in front of the computer. It is preferable to work in short intervals, until more control over the mouse is acquired. In the adaptation phase, it is required to have some patience; otherwise, a user will end up discouraged and put the application down. After several sessions, users begins to slowly control with more precision the cursor displacement and clicks.

The compact native keyboard provided by the software should really be replaced with a "bigger" version, in which the letters and numbers should have their own key. It was also suggested by the users to create a new virtual keyboard with customizable keys, for example, having keys that are configured by the user to play back voice messages, by making use of associated audio files or text-to-speech synthesized voice. The functionality of automatic word completion should also be considered for the next versions.

Voice commands were not covered in the initial application goal, as they appeared as an extra during development. However, multiple users came to prove it to be a functional part of great importance and should inclusively be improved in future work. In those future releases, the list of commands should extend and facilitate for example, the vertical and horizontal scroll of web pages.

There is yet a long way to go in the application improvements, nevertheless, for now, it is an open source low-cost tool (≤70€ for the necessary hardware), that already shows good results and is an alternative to the commercial ones. Plus, being an open-source tool, available in a public git repository at [https://github.com/MigMart/Usertracking], we can expect several improvements made by a large community that wants to associate with it.

Acknowledgements

This work was supported by the Portuguese Foundation for Science and Technology (FCT) project LARSyS (UID/EEA/50009/2013), CIAC (PEstOE/EAT/UI4019/2013), PRHOLO QREN I&DT n. 33845, POPH and FEDER. We would like also to thanks our college Prof. Pedro Cardoso for the valuable comments, suggestions and revisions.

References

1. Nanni L, Lumini A, Dominio F, Zanuttigh P. Effective and precise face detection based on color and depth data. *Applied Computing and Informatics*; 2014, 10(1-2):1-13.
2. Wang YX, Lo LY, Hu MC. Eat as much as you can: a kinect-based facial rehabilitation game based on mouth and tongue movements. *In Proc. of the ACM Int. Conf. on Multimedia*; 2014. p. 743-744.
3. Xiong X, Cai Q, Liu Z, Zhang Z. Eye gaze tracking using an RGBD camera: a comparison with a RGB solution. *In Proc. of the 2014 ACM Int. Joint Conf. on Pervasive and Ubiquitous Computing: Adjunct Publication*; 2014, p. 1113-1121.
4. Malawski F., Kwolek B, Sako S. Using Kinect for facial expression recognition under varying poses and illumination. *Active Media Technology, Springer International Publishing*; 2014, p. 395-406.
5. Han B, Lee S, Yang HS. Head pose estimation using image abstraction and local directional quaternary patterns for multiclass classification. *Pattern Recognition Letters*, 2014, 45:145-153.
6. Kondori FA, Yousefi S, Li H. Direct three-dimensional head pose estimation from Kinect-type sensors. *Electronics Letters*; 2014, 50(4):268-270.
7. Chen YL, Kuo TS, Chang WH, Lai JS., A novel position sensors-controlled computer mouse for the disabled. *In Proc. of the 22nd Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*; 2000, 3:2263-2266.
8. Rajaee S. Web browser software with single switch interface for PDAs or computers. *In Proc. of the IEEE 30th Annual Northeast Bioengineering Conf.*; 2004, p. 253- 254.
9. Connor C, Yu E, Magee J, Cansizoglu E, Epstein S, Betke M. Movement and recovery analysis of a Mouse-Replacement interface for users with severe disabilities. *In Proc. 5th Int. Conf. UAHCI 2009 (held as part of HCI Int. 2009)*; 2009, p.493-502.
10. CameraMouse, http://www.cameramouse.org, accessed 10/03/2014.
11. Dynavox EyeMax System, http://www.dynavoxtech.com/products/eyemax/, accessed 10/03/2014.
12. Toby-Churchill, Dynavox EyeMax System, http://www.toby-churchill.com/products/dynavox-eyemax-system/, accessed 10/03/2014.
13. Quha Zono Mouse, http://www.quha.com/products-2/zono/, accessed 10/03/2014.
14. myGaze Eye Tracker, http://www.mygaze.com/products/mygaze-eye-tracker/, accessed 10/03/2014.
15. KinesicMouse, http://kinesicmouse.com/, accessed 10/03/2014.
16. Alves R, Madeira M, Ferrer J, Costa S, Lopes D, Mendes da Silva B, Sousa L, Martins J, Rodrigues JMF. Fátima revisited: An interactive installation. *In Proc. Int. Multidisciplinary Scientific Conf. on Social Sciences and Arts*, Varna, Bulgaria, 2014, p. 141-148
17. Alves R, Sousa L, Negrier A, Rodrigues JMF, Cardoso PJS, Monteiro J, Gomes M, Bica P. PRHOLO: Interactive Holographic Public Relations, *submitted to 3rd Int. Conf. on Advances in Computing, Communication and Information Technology*, Birmingham, UK, 2015.
18. Kinect for Windows Sensor, Components and Specifications, http://msdn.microsoft.com/en-us/library/jj131033.aspx, accessed 10/03/2014.
19. Hejlsberg A, Wiltamuth S, Golde, P. *The C# programming language*. Adobe Press, 2006.
20. Kinect for windows, Technical documentation and tools, http://goo.gl/zkXhgp, 15/04/2014.
21. Zhou M, Liang L, Sun J, Wang Y, AAM based face tracking with temporal matching and face segmentation. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*; 2010, p.701-708.
22. Cao Z, Yin Q, Tang X, Sun J. Face recognition with learning-based descriptor. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*. 2010, p.2707-2714.
23. Yin Q, Tang X, Sun J. An associate-predict model for face recognition. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*; 2011, p.497-504.
24. CANDIDE – a parametrized face, http://www.icg.isy.liu.se/candide/, accessed 10/07/2014.
25. Microsoft Developer Network, Face Tracking, http://msdn.microsoft.com/enus/library/jj130970.aspx, accessed 20/03/2014.
26. How to: Simulate Mouse and Keyboard Events in Code, http://goo.gl/3nVCkF, accessed 29/12/2014.
27. Microsoft Speech Platform SDK 11 Documentation, http://goo.gl/qqJhHk, accessed 29/12/2014.
28. Abhijit, J. *Kinect for Windows SDK Programming Guide*. Packt Publishing Ltd, 2012.